Block ciphers: DES and AES

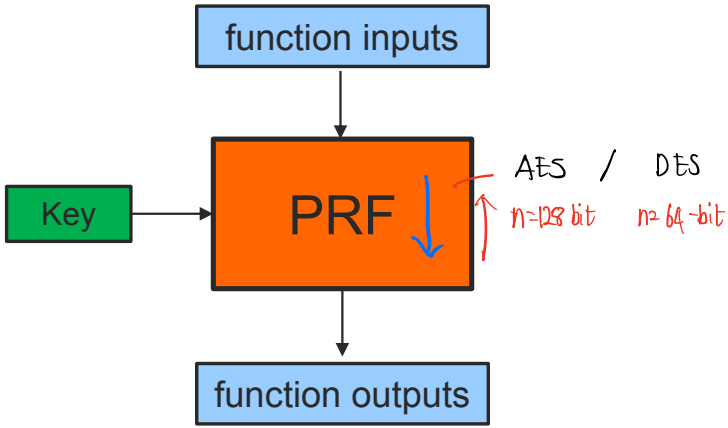
3DES

Jong Hwan Park

Pseudo-Random Function(PRF)

- An efficient, keyed function as $F : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$

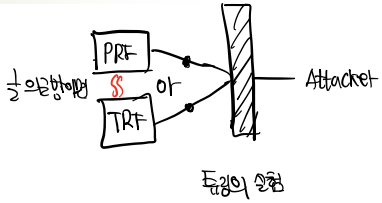
실제로는 Input과 output의 length는 동일.

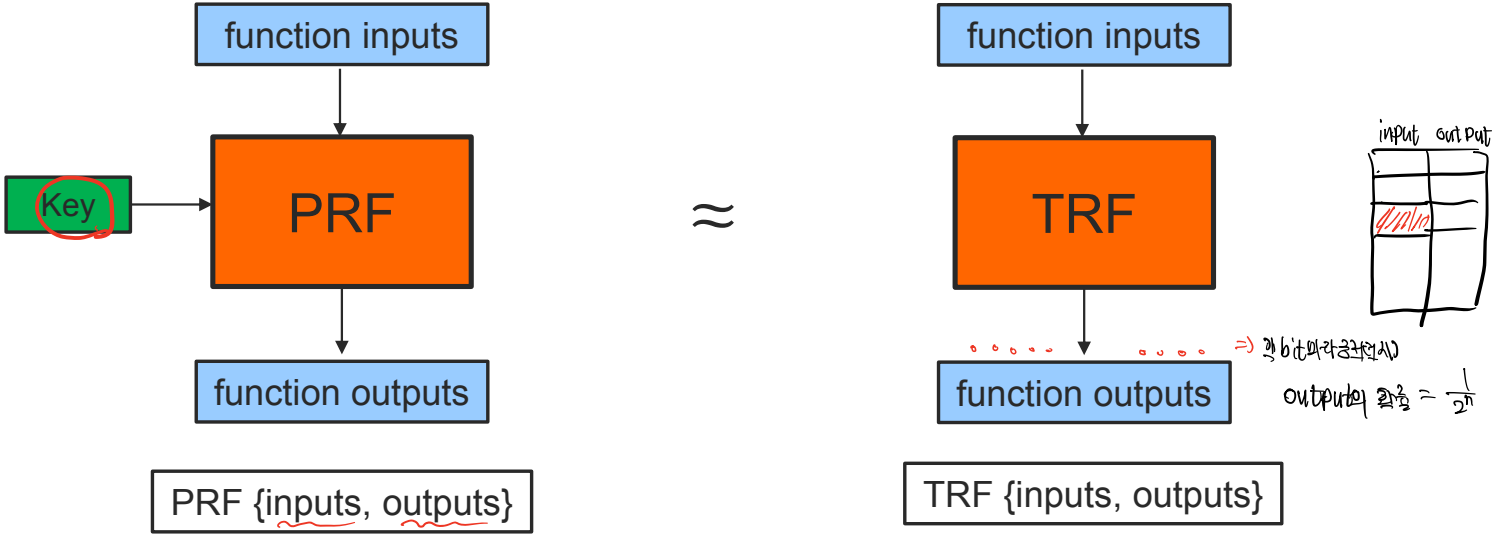


- Function evaluation (F_K) is efficiently computable given k
- F_K is length-preserving when m=n
- Function inversion (F_K^{-1}) is not considered

Security of PRFs

- Indistinguishability between PRF and TRF
  - TRF – True Random Function
  - Security depends on how good PRF outputs look random

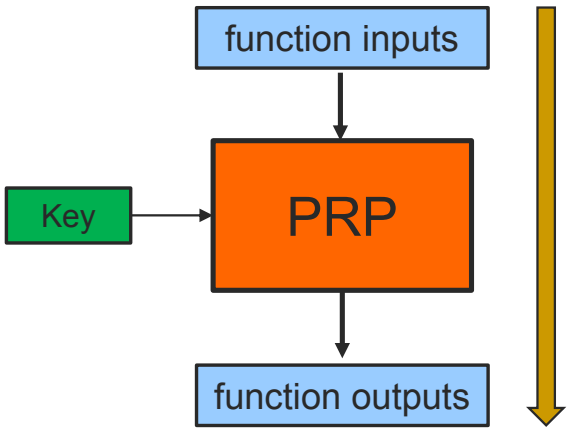


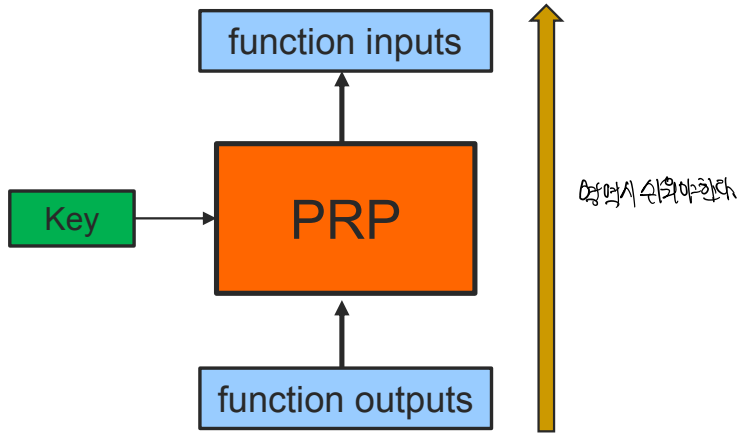


  - Attacker's goal is to distinguish if {inputs, outputs} is from F_K or f
  - Randomness check tests for PRF

Pseudo-Random Permutation(PRP)

- An efficient, keyed permutation (one to one mapping) as $F : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$

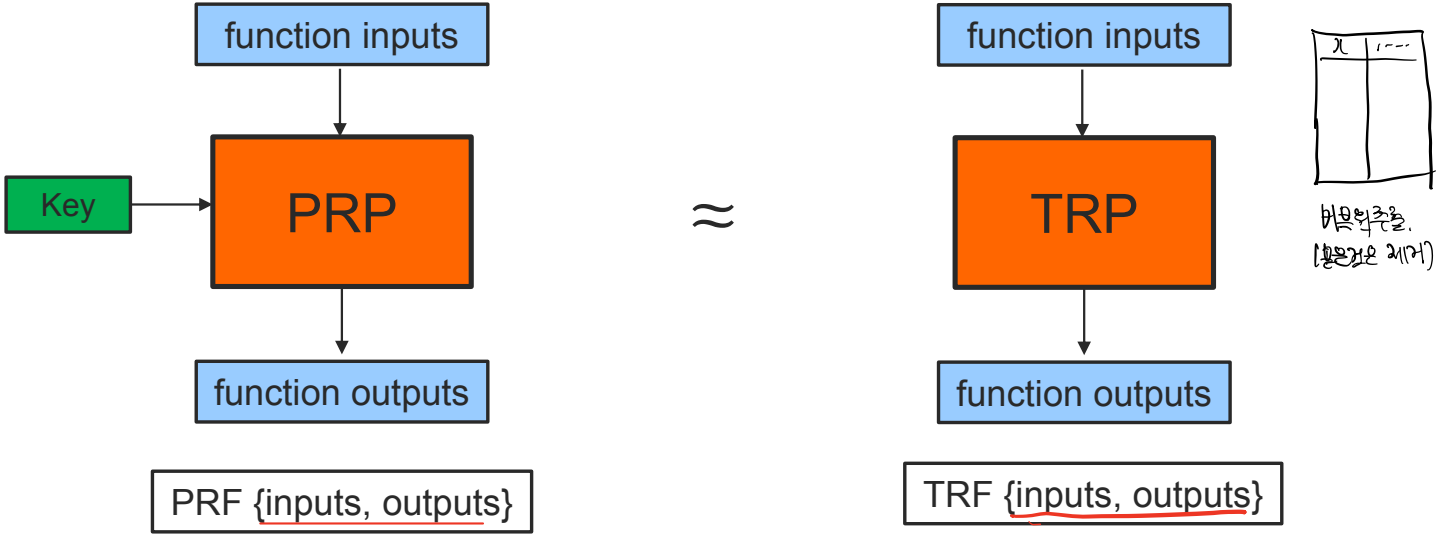




- Function evaluation (F_K) is efficiently computable given k
- F_K should be length-preserving
- Function inversion (F_K^{-1}) is also efficiently computable

Security of PRPs

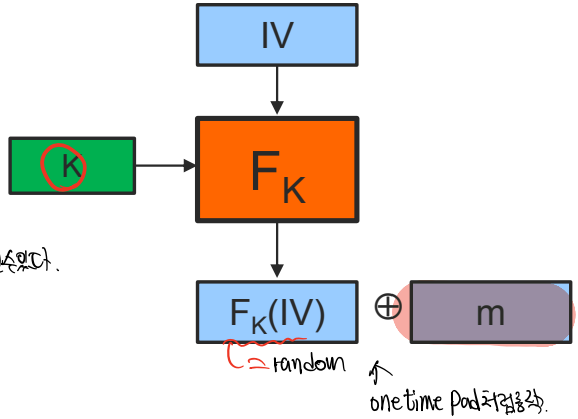
- Indistinguishability between PRP and TRP
  - TRP – True Random Permutation
  - Security depends on how good PRP outputs look random



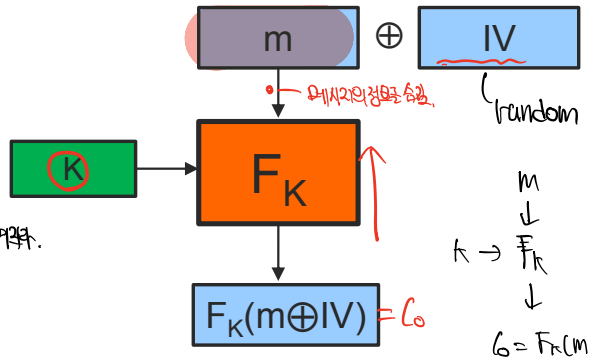
  - Attacker's goal is to distinguish if {inputs, outputs} is from F_K or f
  - Randomness check tests for PRP

Basic Encryption using PRF and PRP

- Using PRF
  - Enc: $C = [IV, F_K(IV) \oplus m]$
  - Dec: $m = C_0 \oplus F_K(IV)$
  - Why is it secure?
  - Basis on CTR mode

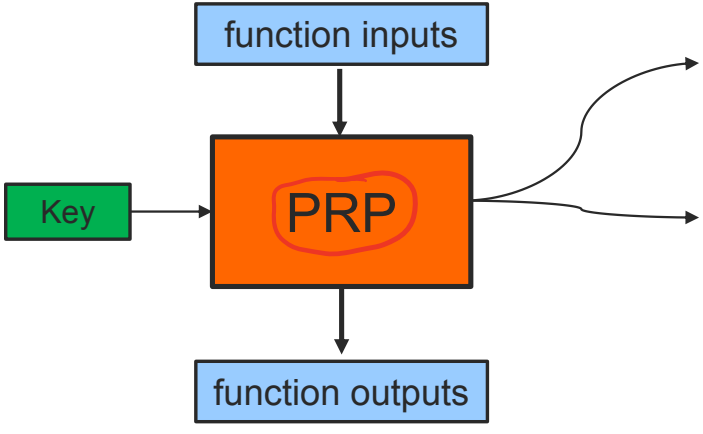


- Using PRP
  - Enc: $C = [IV, F_K(IV \oplus m)]$
  - Dec: $m = IV \oplus F_K^{-1}(C_0)$
  - Why is it secure?
  - Basis on CBC mode



If |m| is longer than n bits, need *mode of operation*

Two Approaches to PRPs



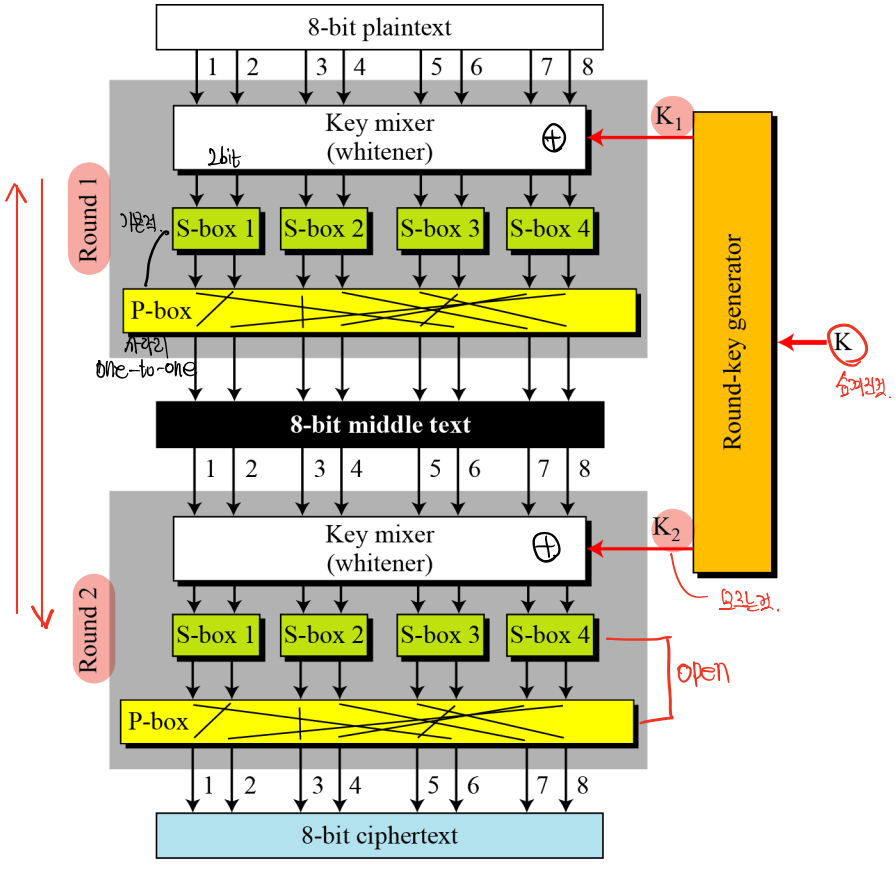
- Substitution-Permutation Network
  - AES
- Feistel Network
  - DES

- Often, called '<u>block cipher</u>' (as an efficient, keyed permutation)
- In practice, block length is n = <u>64</u> (3DES) and n=<u>128</u> (AES)
  - n is the size of the bitlength of output/input
- F_K is denoted as 'encryption', and F_K^{-1} as '<u>decryption</u>'

SPN for constructing block cipher

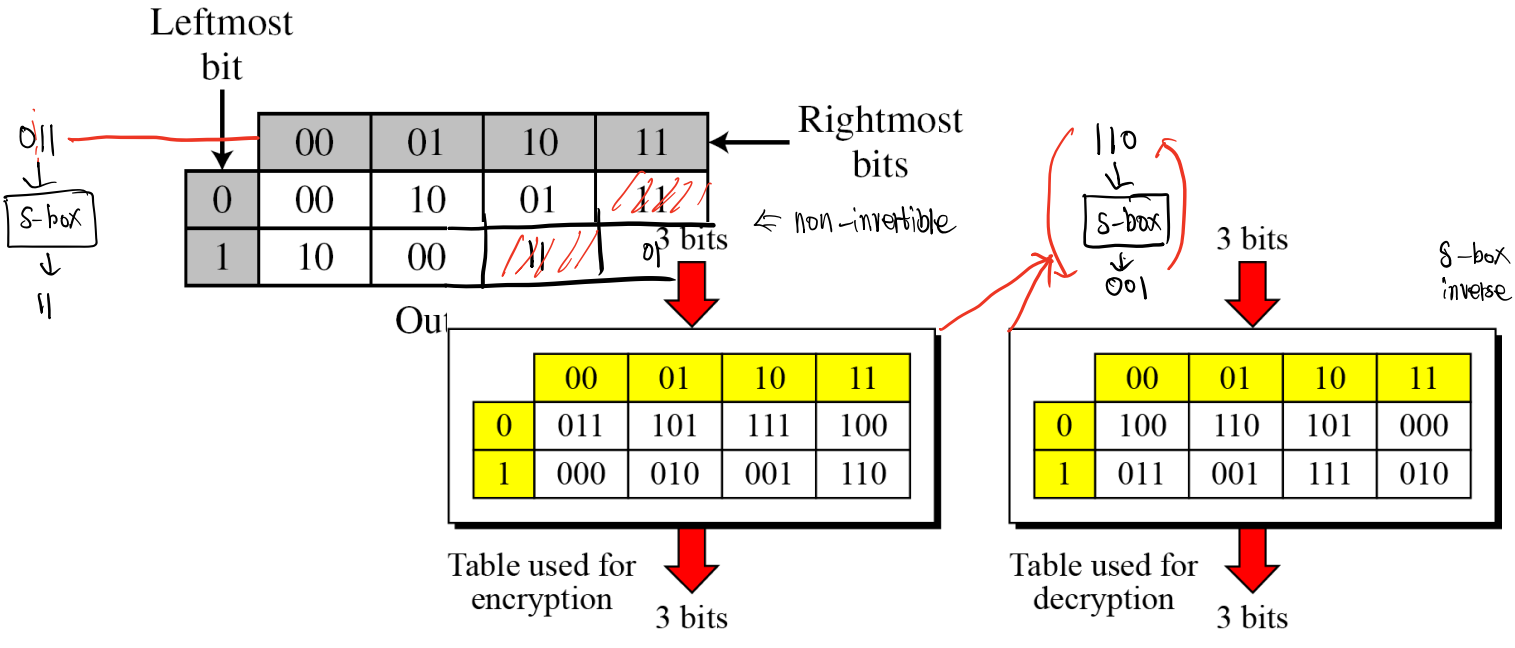
- Substitution-Permutation Network (SPN)
  - Substitution (S-box)
  - Permutation (P-box)
  - Rounds, and key schedule
- Shannon's idea:

  "*Weak* tools like substitution and permutation can lead to a *secure* cipher if the number of rounds increases"



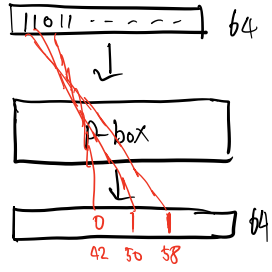
Components – Substitution (S-box)

- An S-box is an $m \times n$ substitution unit, where m and n are not necessarily the same
- Example: invertible vs. non-invertible



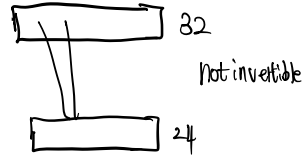
Components – Permutation (P-box)

- Permutation: straight, compression, expansion
  - Straight P-box is only invertible
  - Example: (a) 64 × 64, (b) 32 × 24, (c) 12 × 16



58	50	42	34	26	18	10	02	60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06	64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01	59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05	63	55	47	39	31	23	15	07

01	02	03	21	22	26	27	28	29	13	14	17
18	19	20	04	05	06	10	11	12	30	31	32



01	09	10	11	12	01	02	03	03	04	05	06	07	08	09	12

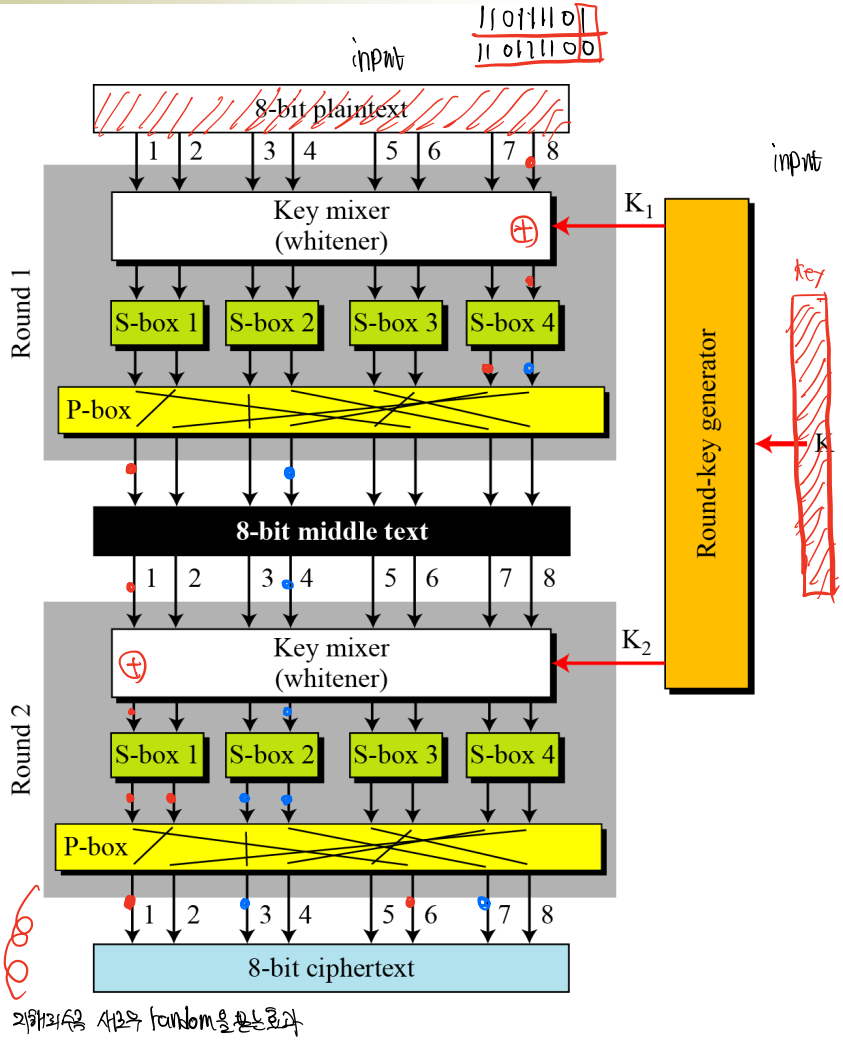
Design Principle of SPN

- Invertibility of S-boxes
- Avalanche effect:

  Changing a single bit of the input affects every bit of output

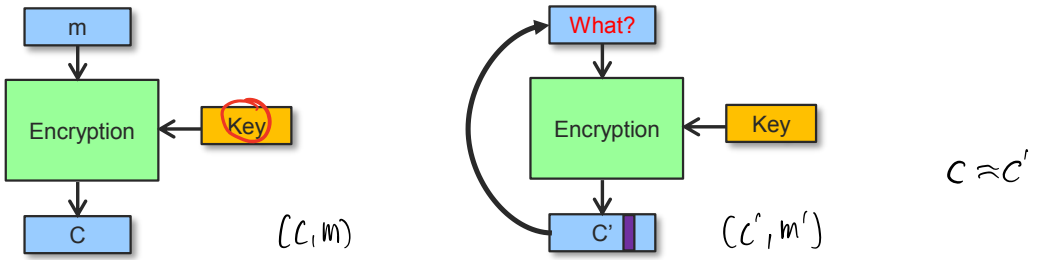
  → S-box changes at least two bits of output

  → P-box spreads output of any S-boxes into different S-boxes

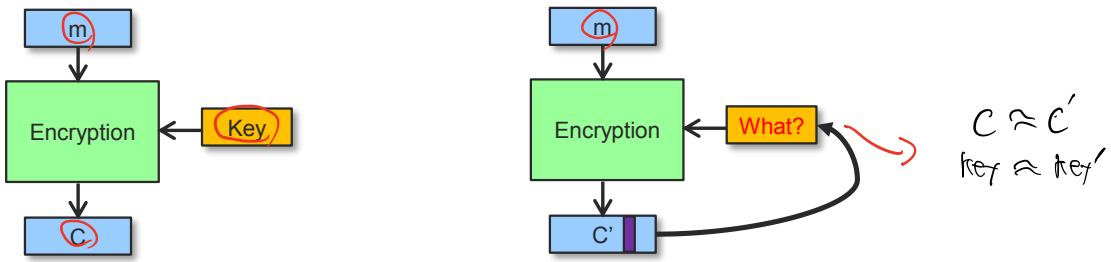


Avalanche Effect (1)

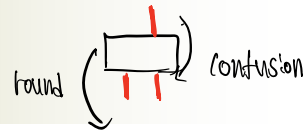
- What if avalanche effect is not provided?
- From the relation between input and output:
  - Given (C, m) and (C', ?) under the same key K, where C ~ C'



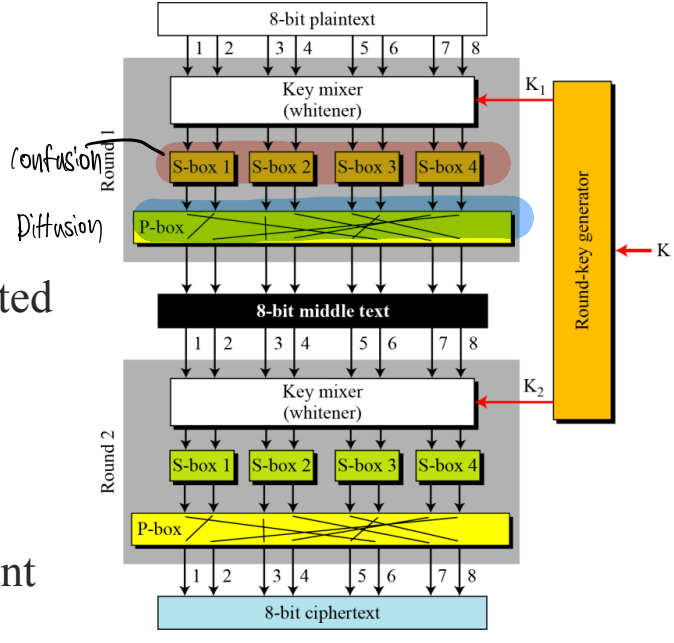
- From the relation between ciphertext and key:
  - Given (C, m, K) and (C', m, ?) , where C ~ C'



Confusion & Diffusion of SPN

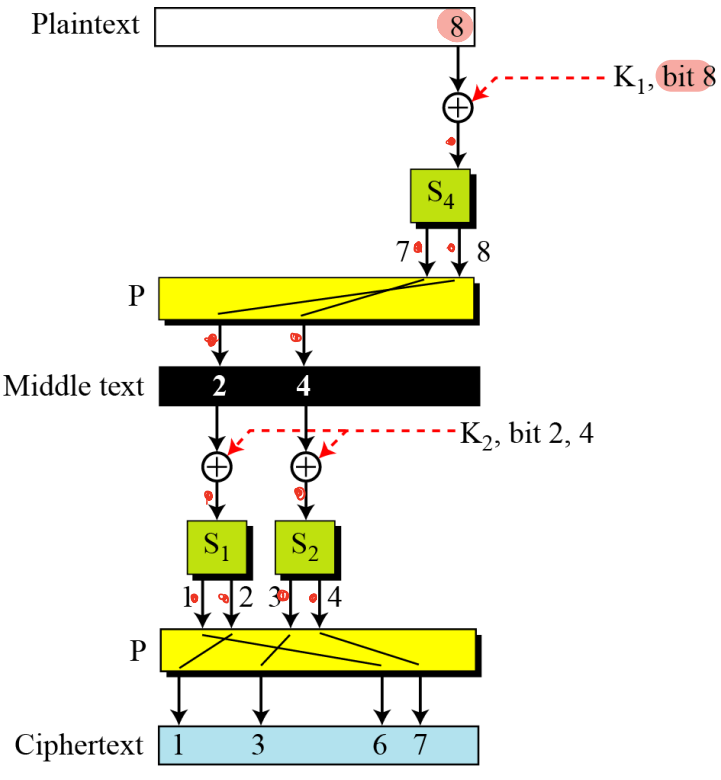


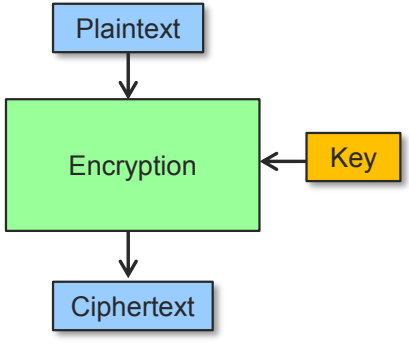
- Confusion (혼돈)
  - Introduce smaller random-looking permutations $\{f_i\}$ → S-box
- Diffusion (확산)
  - The bits of confusion output are permuted or mixed → P-box
- As the number of rounds increases
  - Small changes to input have a significant effect on the output
    → Would expect a random permutation



Avalanche Effect (2)

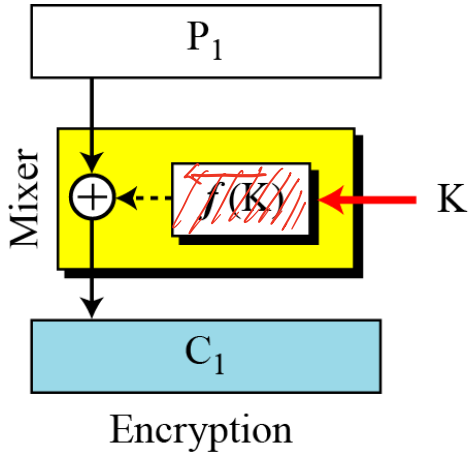
- Due to the avalanche effect, a single bit change of input (key or plaintext) is propagated to all the bits of output

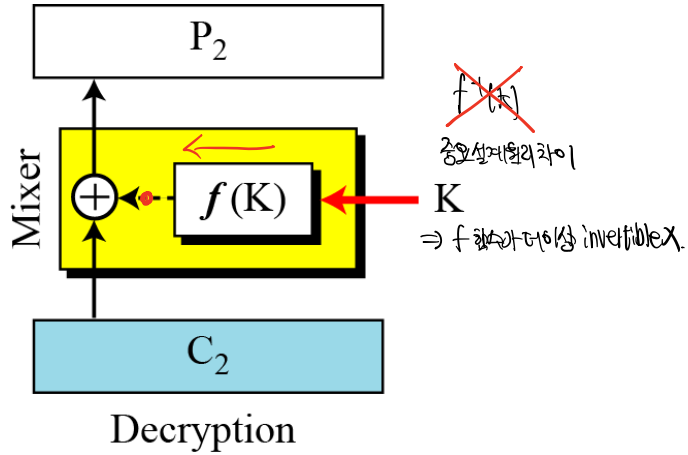




Feistel Networks for constructing block cipher

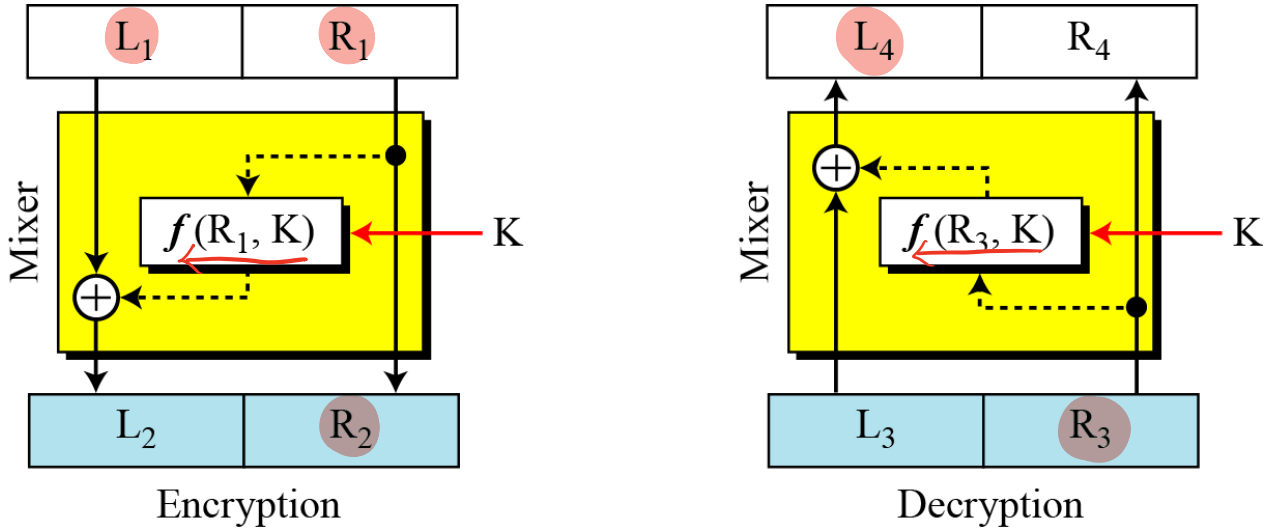
- S-boxes, P-boxes, rounds, and key schedule are the same
- Advantage of Feistel networks over SPN
  - S-boxes no longer need to be invertible
  - If S-boxes (in f(K)) are non-invertible, how can we decrypt ciphertexts?





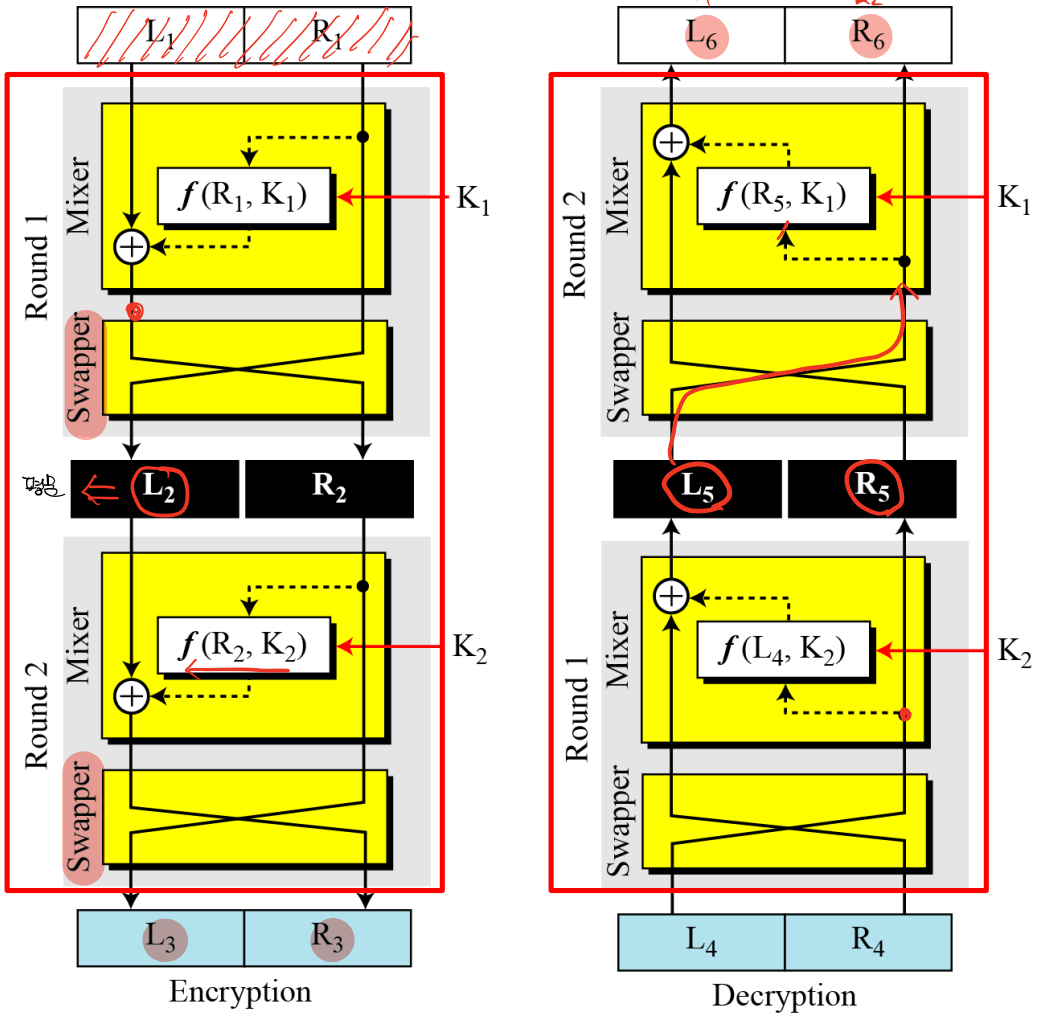
Improvement of Feistel Networks

- Divide plaintexts and ciphertexts into two equal-length blocks



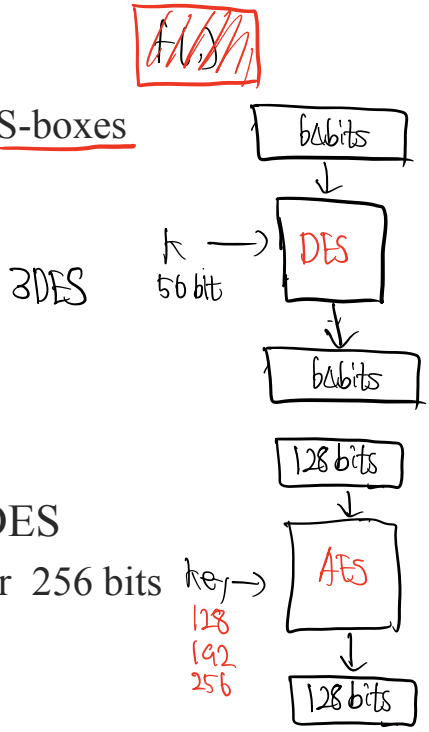
  - Want message (as well as K) to be inserted into $f(R_j, K)$

Feistel Cipher with two rounds



Practical Constructions of Block Cipher

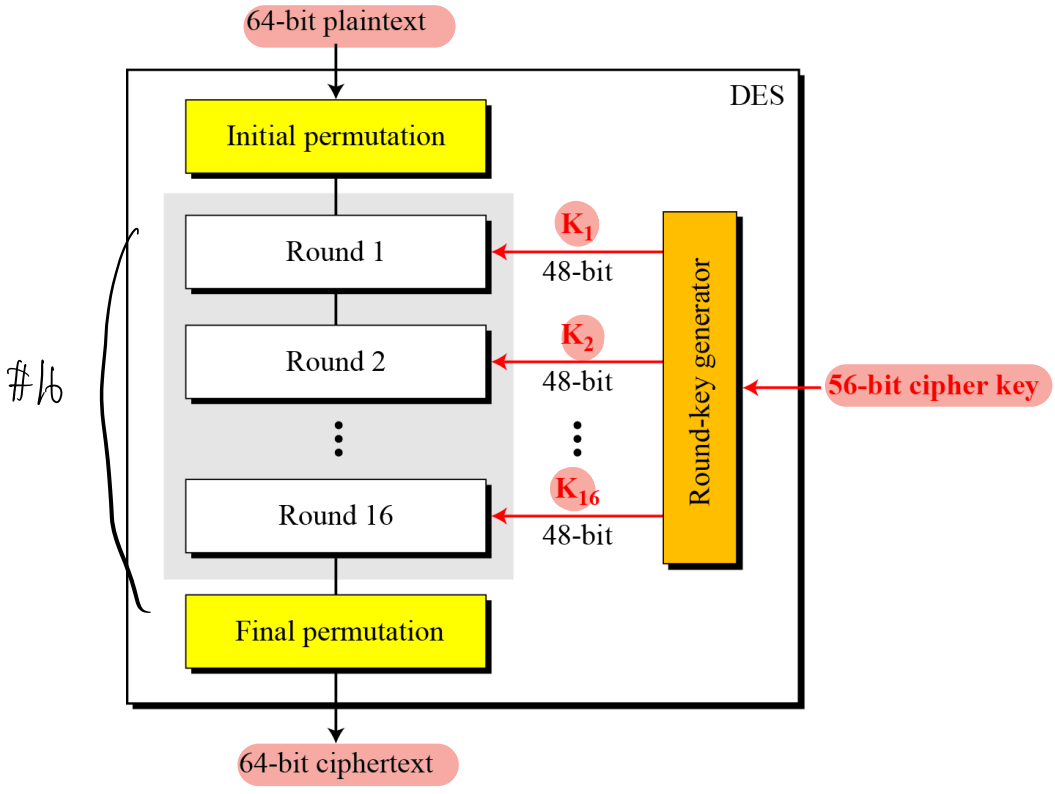
- Data Encryption Standard (DES)
  - Based on Feistel Networks
  - In 1973, NBS asks for block cipher proposals
    - IBM submits variant of Lucifer
  - In 1976, NBS adopts DES as a federal standard
    - Key length = 56 bits & hidden design principle for S-boxes
  - In 1999, Triple-DES was published as a standard
    - Key length = 2 × 56 or 3 × 56 bits

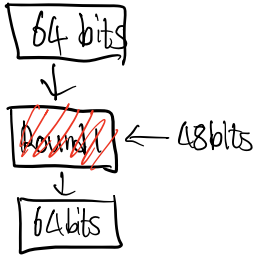


- Advanced Encryption Standard (AES)
  - Based on similar fashion of SPN
  - In 1999, NIST was looking for a replacement for DES
    - Block size = 128 bits & key length = 128 or 192 or 256 bits
  - In 2001, Rijndael was published as AES

DES Structure (1)

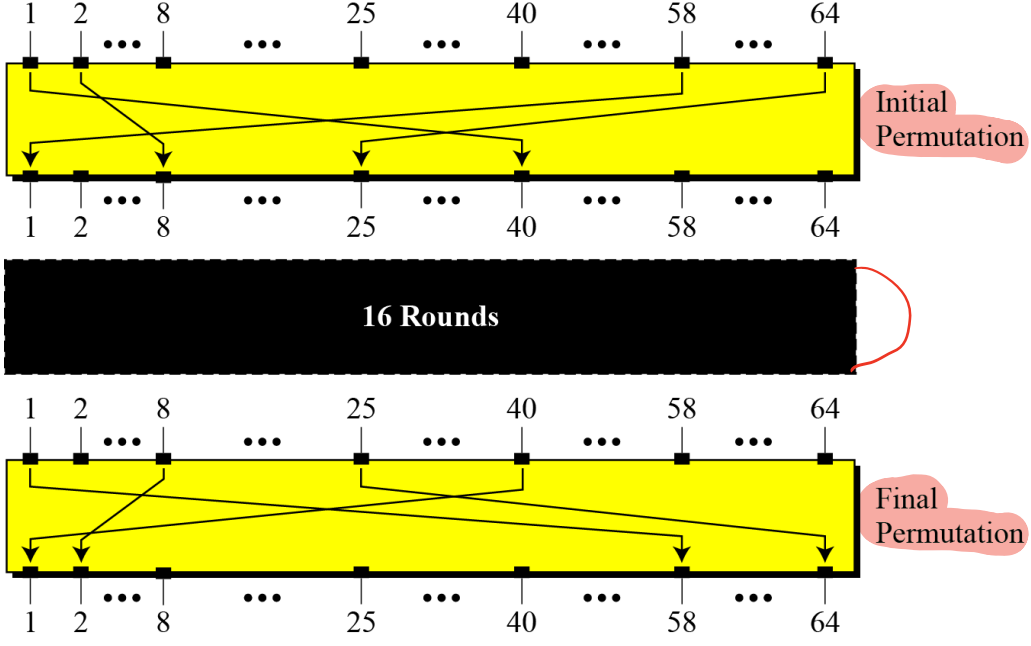
- General structure
  - How to decrypt ?





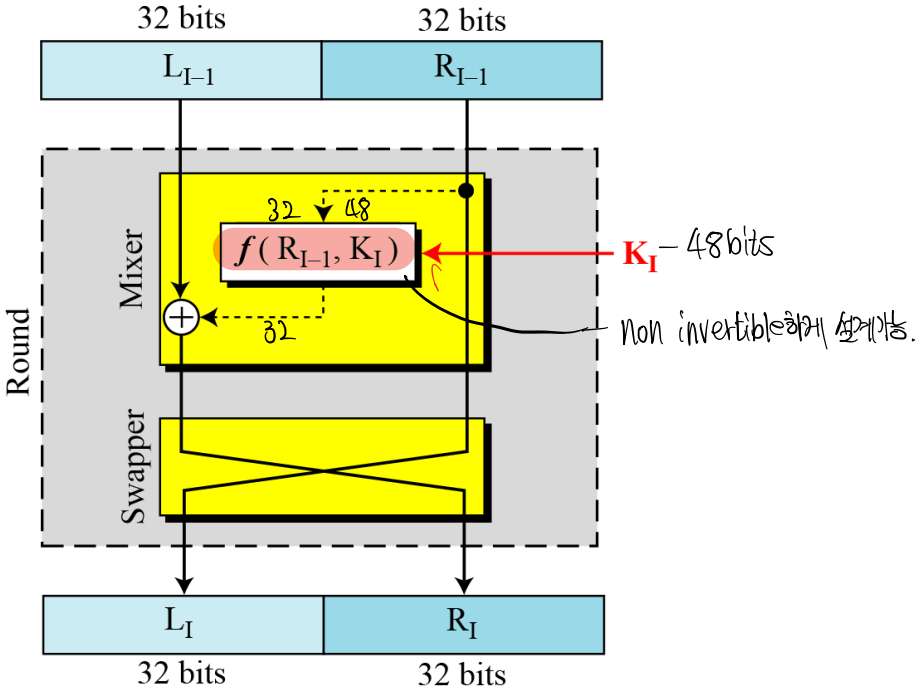
DES Structure (2)

- Initial and final permutation
  - Keyless permutation and inverse of each other
  - No cryptographic significance in DES



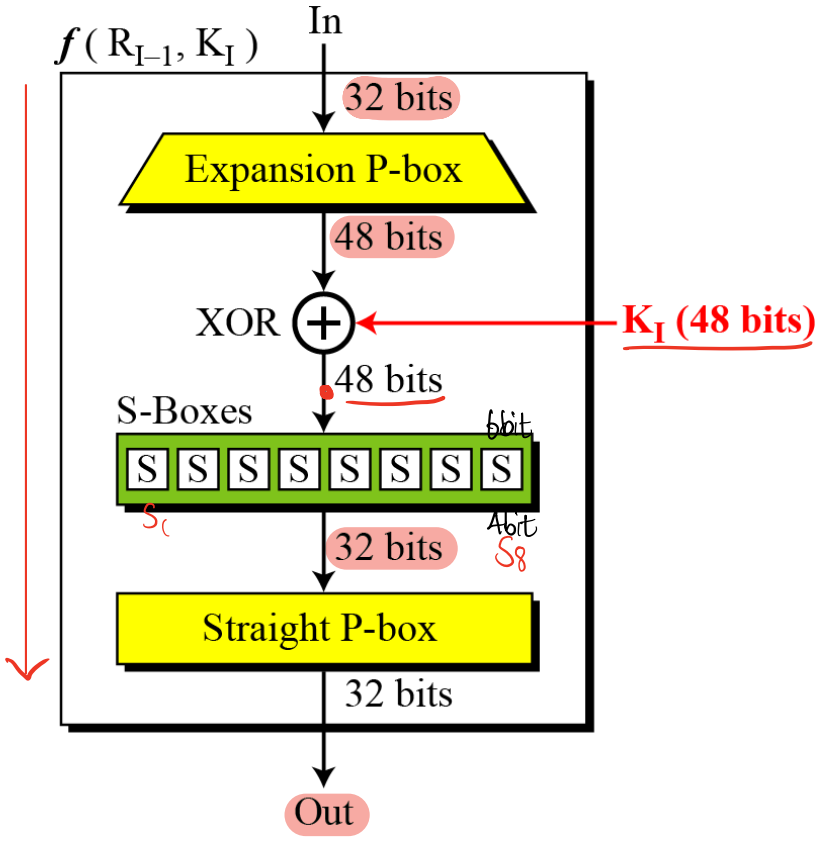
DES Structure (3)

- A round in DES (encryption site)
  - DES uses 16 rounds
  - All *non-invertible* elements are collected inside $f(R_{I-1}, K_I)$ function



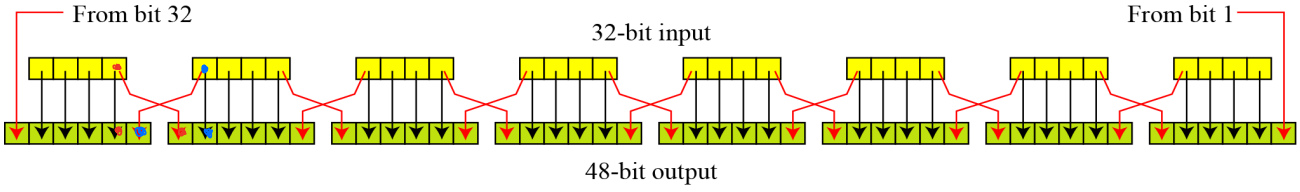
DES Structure (4)

- DES function – the heard of DES f function

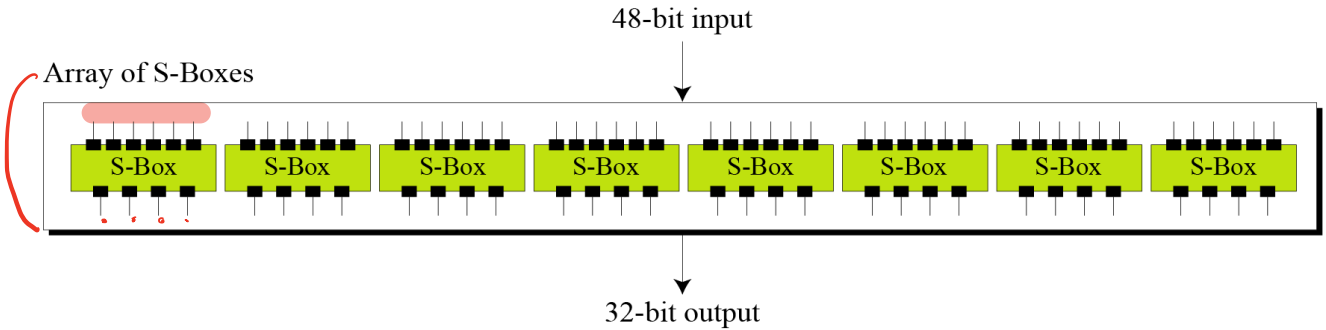


DES Structure (5)

- Expansion P-box



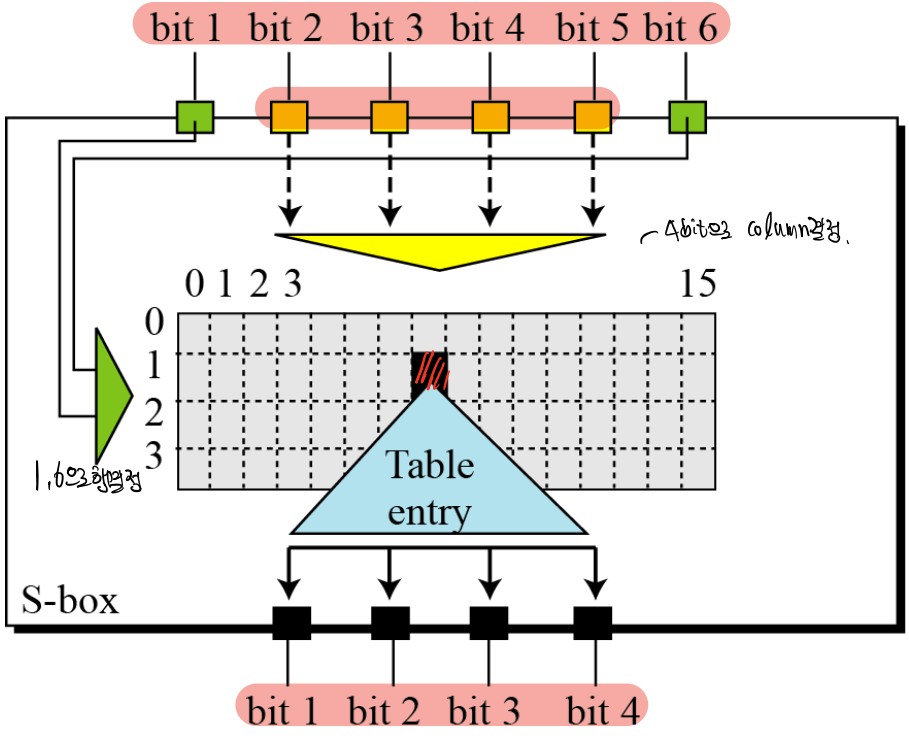
- XORed with 48 bits subkey
- S-boxes (6 bits to 4 bits)



- Straight permutation (32 bits)

DES Structure (6)

- S-box rule, implemented as look-up table



DES Structure (7)

- S-box 1
  - The input of S-box 1 is 101011. What is the output?

101011 → S-box → 1001

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

1001

  - What is the reason that the numbers in S-box are arranged in that way?
    - Later, IBM designers mentioned that S-boxes are designed to prevent *differential* cryptanalysis (DC)

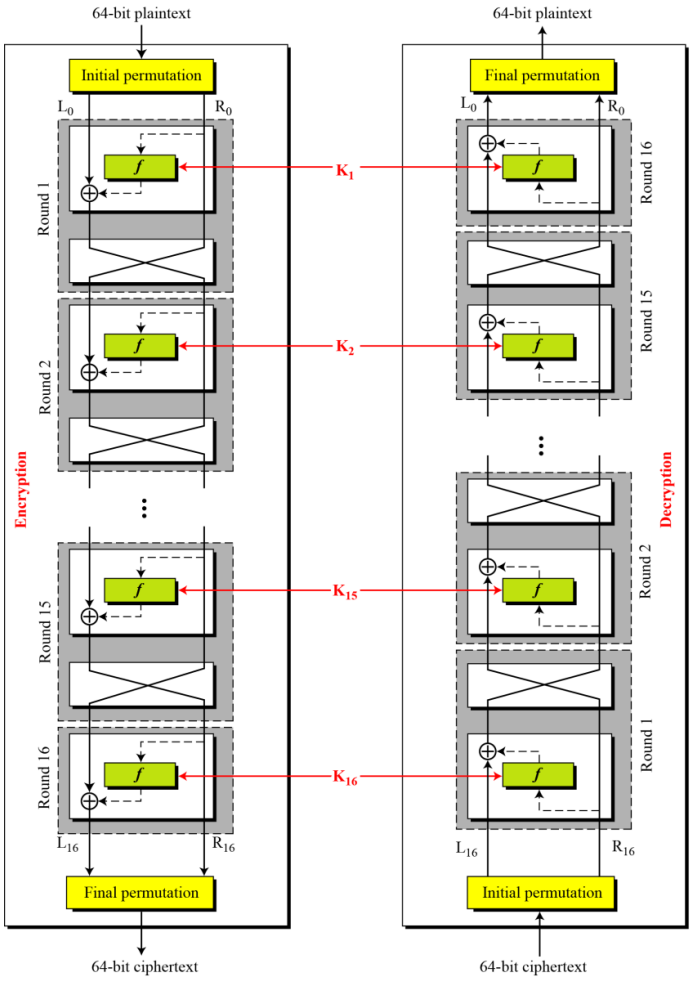
시사점 1. 이 배열은 바뀌면 안된다. ⇒ 바로 깨진다

2. 그 당시 개발자들이 공격 방법인 DC를 이미 알고 있었다.

3. DES 설계한 사람들은 후에 학자들이 DC라는 공격으로 분석할 것을 알고 있었다.

DES Structure (8)

- DES cipher and decipher

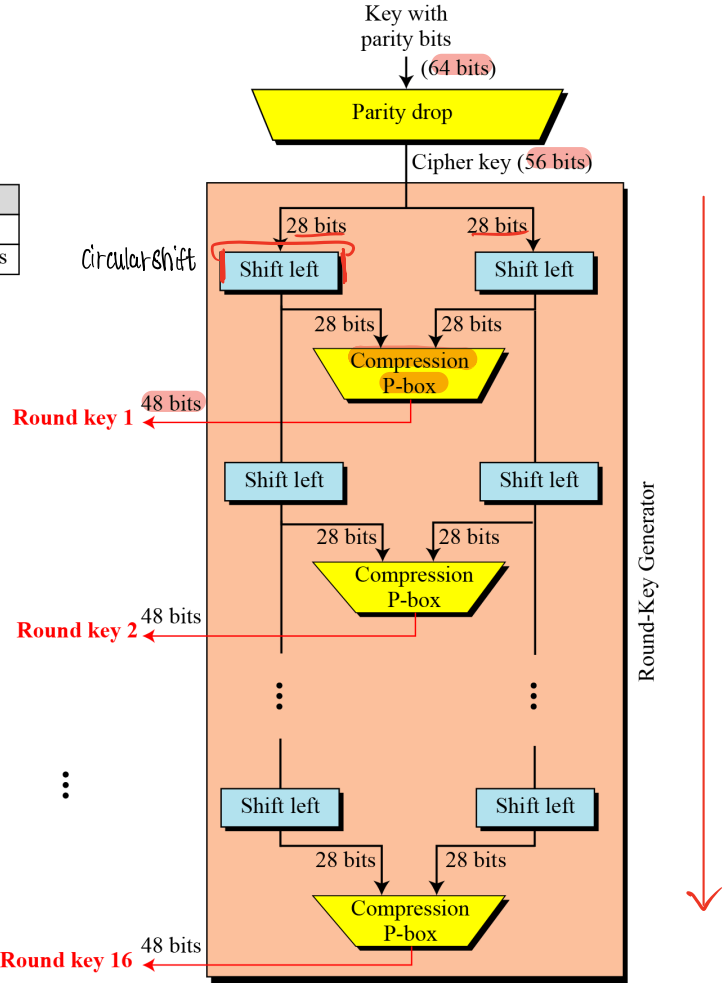


DES Structure (9)

- Key generation

Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

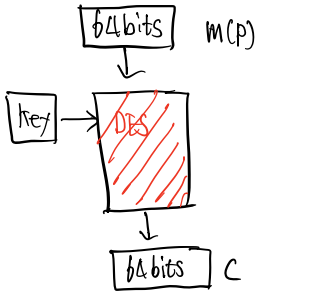


DES properties

- Avalanche effect
  - Encrypt plaintexts that differ only in a single bit

Plaintext: 0000000000000000 Key: 22234512987ABB23
Ciphertext: 4789FD476E82A5F1

Plaintext: 0000000000000001 Key: 22234512987ABB23
Ciphertext: 0A4ED5C15A63FEA3



  - Number of bit differences in rounds

Rounds	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Bit differences	1	6	20	29	30	33	32	29	32	39	33	28	30	31	30	29

보수 1101 = 0010

- Key complement effect
  - $C = Enc(K, P) \Leftrightarrow \bar{C} = Enc(\bar{K}, \bar{P})$
  - Can use 2^{55} keys (not 2^{56}) to perform exhaustive search for a special case where (C, P) and $(C', \bar{P})$ are given (why?)

given $C(=P)$ and $(C', \bar{P})$,
$K \Rightarrow 2^{56}$
$P = 2^{56}$
K_1 K_2 $K_{\frac{1}{2}}$ $K_{\frac{1}{2}}$
$Enc(K_1, P) = \hat{C}$
$Enc(K_2, P) = C$
→ 1. $\hat{C} = C \rightarrow K = K_1$
2. $\hat{C} \neq C \rightarrow$ • $Enc(\bar{K}_1, \bar{P}) = \bar{\hat{C}}$ (이경우)
1. $\bar{\hat{C}} = C' \rightarrow \bar{K}_1$
2. $\bar{\hat{C}} \neq C' \rightarrow$ ✗
1번의 방법으로 K_1, $\bar{K}_1$ 둘중에서 하나만 찾아내면 됨.
$2^{56}/2 = 2^{55}$

Security of DES: exhaustive search

- Goal: given a few input/output pairs $(m_i, c_i = E(k, m_i))$ i=1,..,3, find the key k
  - Need to compute 2^{56} DES algorithms for 2^{56} possible keys
- DES challenge

msg = "The unknown messages is: XXXX … "

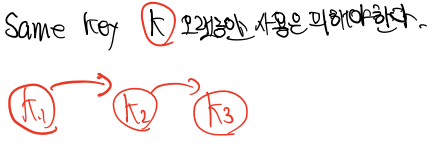
CT = c_1 c_2 c_3 c_4

Goal: find $k \in \{0,1\}^{56}$ s.t. $DES(k, m_i) = c_i$ for i=1,2,3

  - 1997: Internet search – 3 months
  - 1998: dedicated DES-cracking machine – 3 days (250K $)
  - 1999: combined search – 22 hours
  - 2006: COPACOBANA (120 FPGAs) – 7 days (10K $)

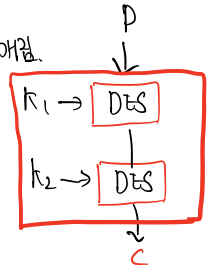
Security of DES

- The best known attack is still an *exhaustive search* via 2^{56} keys
  - But, too short 56-bit key & too short 64-bit block length
    - NSA requested that the key be short enough for them to crack
    - 56-bit ciphers should not be used ! (128-bit key → 2^{72} keys)
  - Differential Cryptanalysis (DC) – in the late 80's
    - Take time 2^{37} keys if 2^{36} pairs of (plaintext, ciphertext) is given to attacker
    - DES S-boxes were designed to resist such DC
      - DES designers were aware of DC and designed DES to thwart the DC
      - Asked by NSA to keep it quiet
  - Linear Cryptanalysis (LC) – in the early 90's
    - LC is new and was not known to DES designers
    - Can be broken if 2^{43} ciphertexts are given to attacker

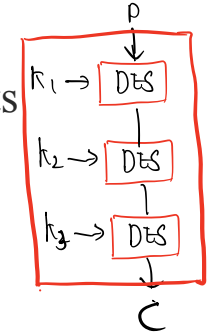


Strengthening DES against exhaustive search

- Two general approaches
  - Internal tampering construction
    - Changing 128-bit master key with a different key schedule
    - Changing S-boxes
  - Block-box construction
    - Treat DES as a block box (use DES as the original form)

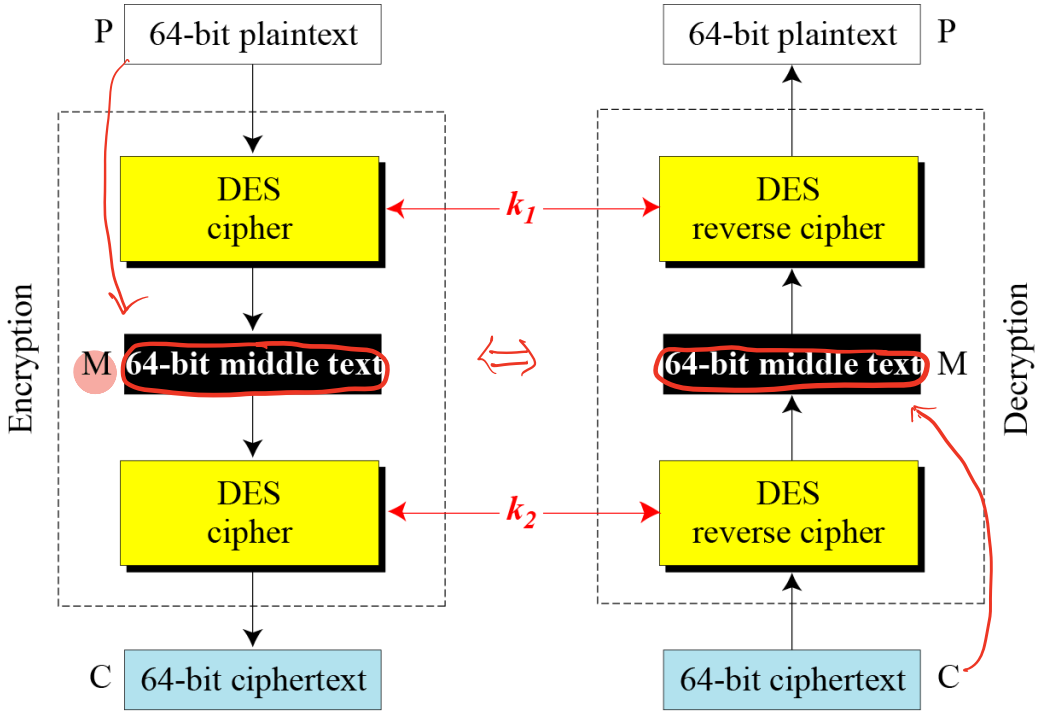


- Double DES:
  - $C = Enc(k_2, Enc(k_1, P))$ with key = (k_1, k_2) of 112 bits
  - Take time about 2^{57} by '*meet-in-the-middle attack*'
- Triple DES:
  - $C = Enc(k_3, Dec(k_2, Enc(k_1, P)))$ with key = (k_1, k_2, k_3) of 168 bits
  - Take time about 2^{112} (no known attack up till now)
  - Widely used as a standard



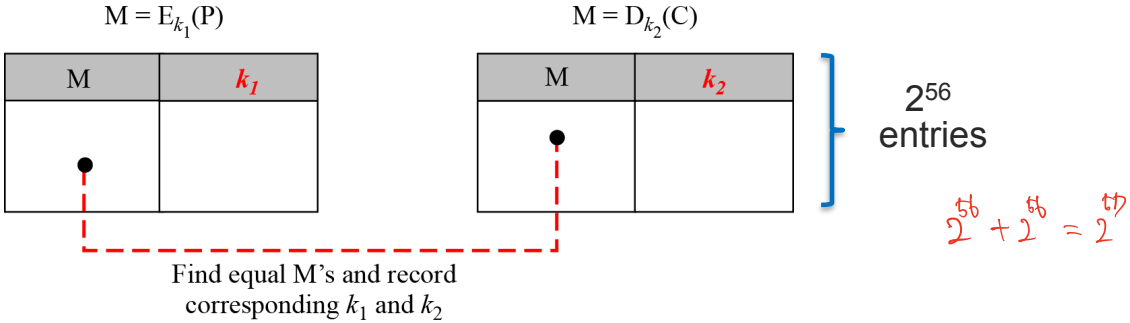
Double DES (1)

- $C = Enc(k_2, Enc(k_1, P))$ with key $= (k_1, k_2)$
  - Given (P, C), observe that $Dec(k_2, C) = Enc(k_1, P)$

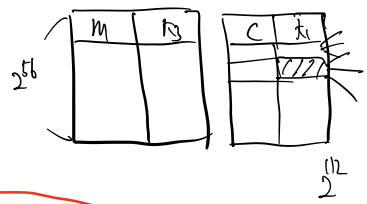


Double DES (2): Meet-in-the-middle attack

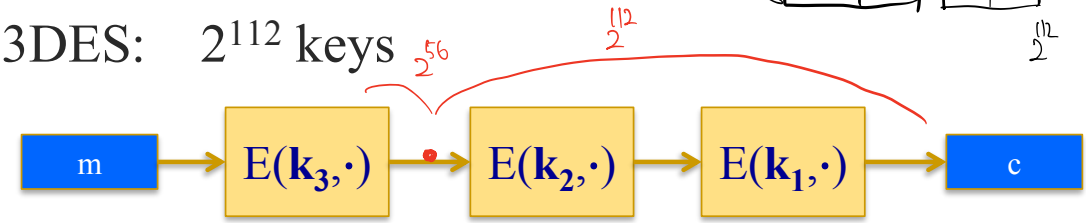
- Given (P, C) to attacker
  - Encrypt the plaintext P using all 2^{56} keys and record all values
  - Decrypt the ciphertext C using all 2^{56} keys and record all values



  - Compare the values in two tables until M is the same
  - Find the key (k_1, k_2) with 2^{57} tries, not with 2^{112}

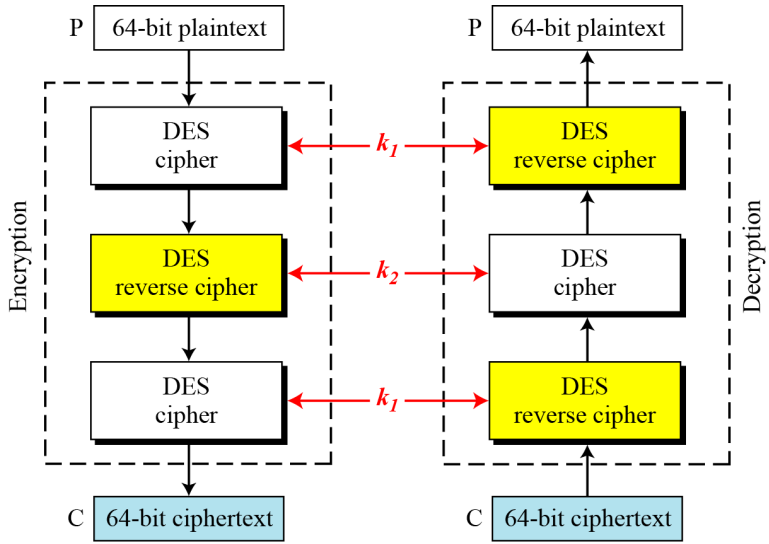


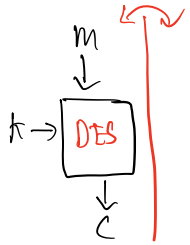
- Same attack on 3DES: 2^{112} keys



Triple DES

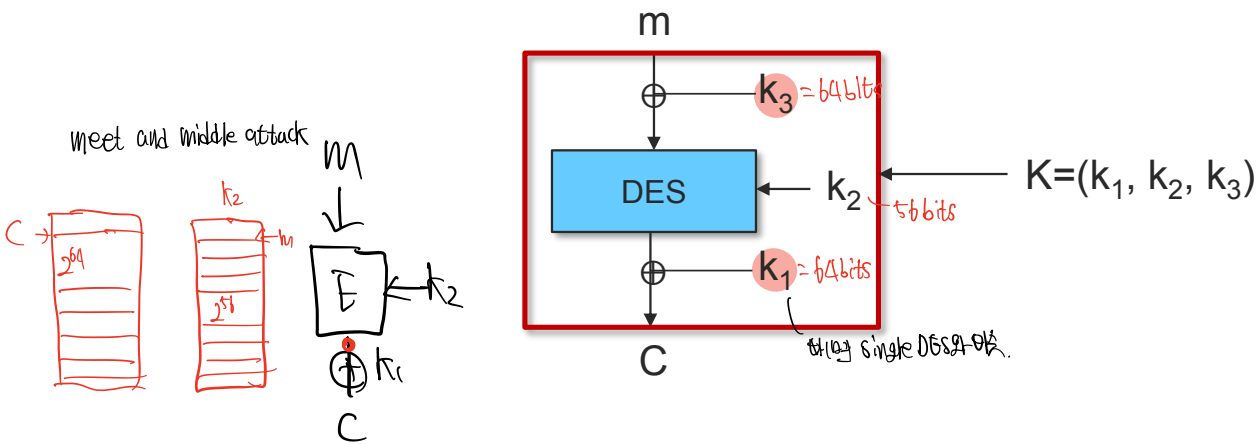
- Two variants
  - C = Enc(k_3, Dec(k_2, Enc(k_1,P))) with three independent keys (k_1, k_2, k_3)
  - C = Enc(k_1, Dec(k_2, Enc(k_1,P))) with two independent keys (k_1, k_2)
- Observe that the middle invocation is Dec
  - Why? If $k_1 = k_2 = k_3$, then it is a single DES (backward compatibility)
  - 1st variant – banking industry / 2nd variant - PGP





Another Method: DESX

- $E : K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ a block cipher
- Define EX as $EX((k_1,k_2,k_3), m) = k_1 \oplus E(k_2, m \oplus k_3)$
  - For DESX: Key length = 64+56+64 = 184 bits
  - … but easy attack in time $2^{64+56} = 2^{120}$



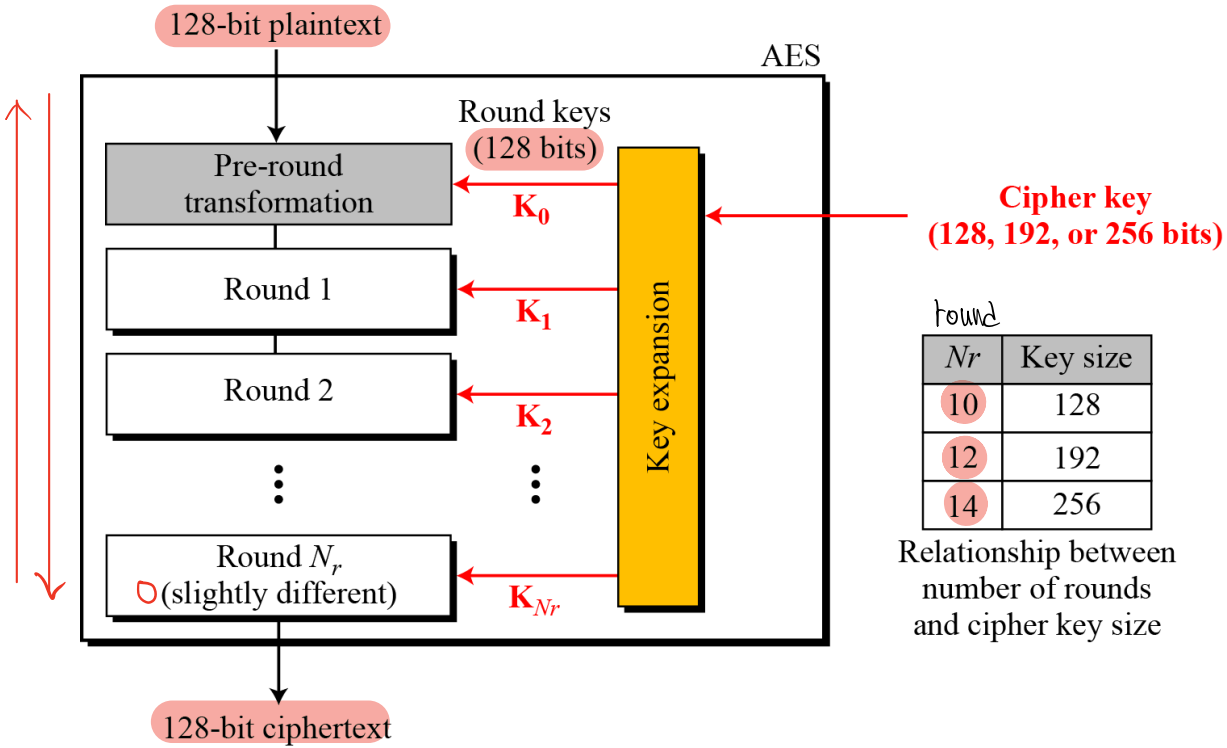
- Note: $k_1 \oplus E(k_2, m)$ and $E(k_2, m \oplus k_1)$ does nothing (why?)
  - Suppose 2^{12} xor operation ~ one DES enc or dec

Advanced Encryption Standard (AES)

- AES process
  - In 1997, NIST started looking for a replacement for DES
  - Selection criteria:
    - Security
    - Cost
    - Implementation
    - Available to the public worldwide
  - In 1998, 15 submissions (five claimed attacks)
  - In 1999, NIST selected 5 candidates
    - MARS, RC6, Rijndael, Serpent, Twofish
  - In 2000, Rijndael was selected as AES (designed in Belgium)
    - Key lengths = 128, 192, 256 bits; block size = 128 bits

General Design of AES encryption

- Based on SPN (not Feistel)

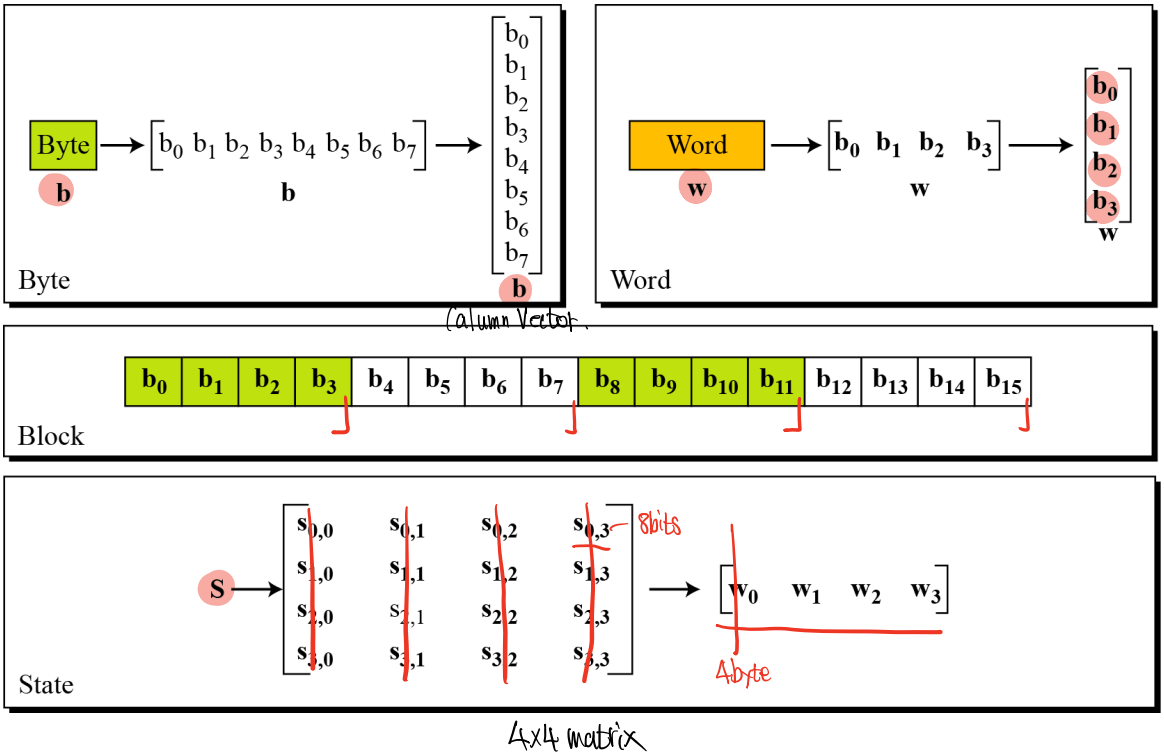


Nr	Key size
10	128
12	192
14	256

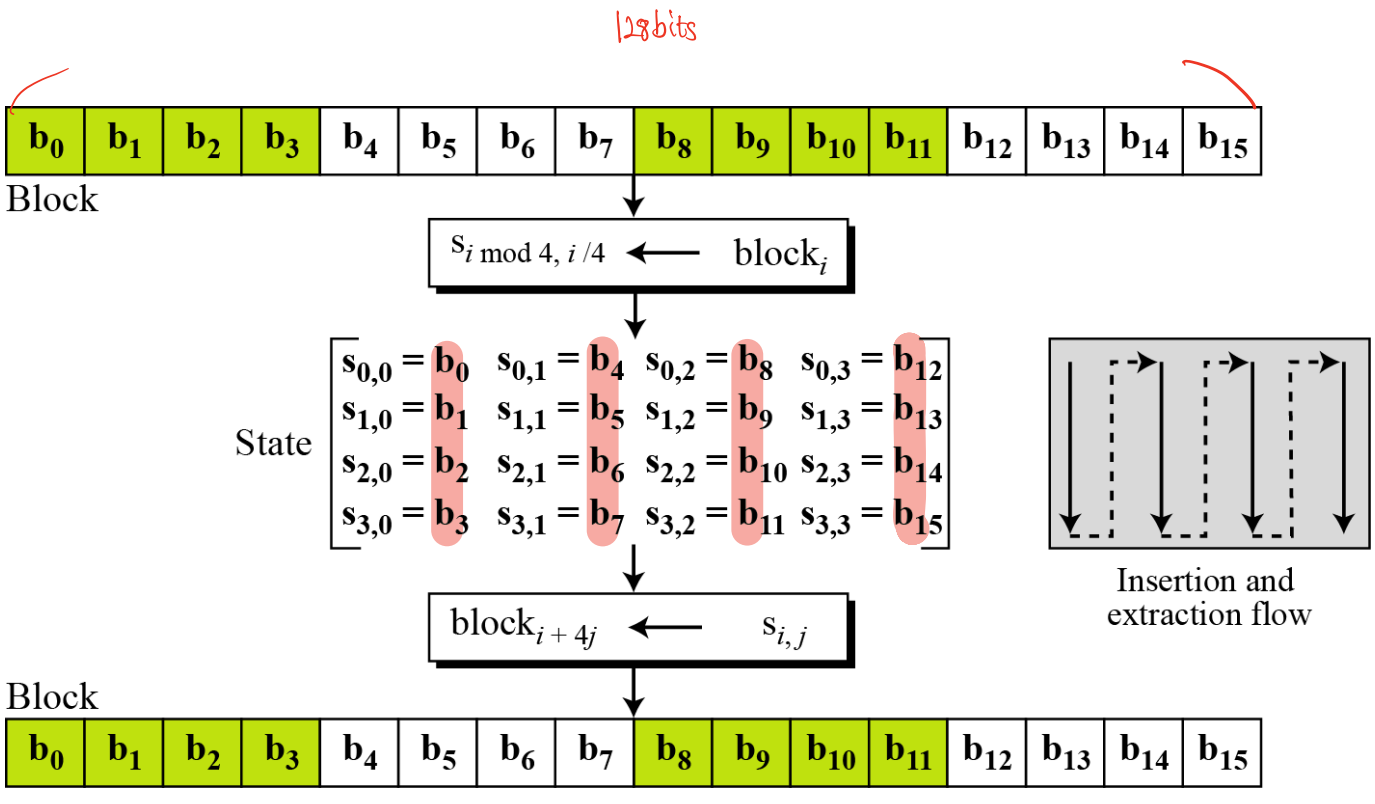
Relationship between number of rounds and cipher key size

Data Units used in AES

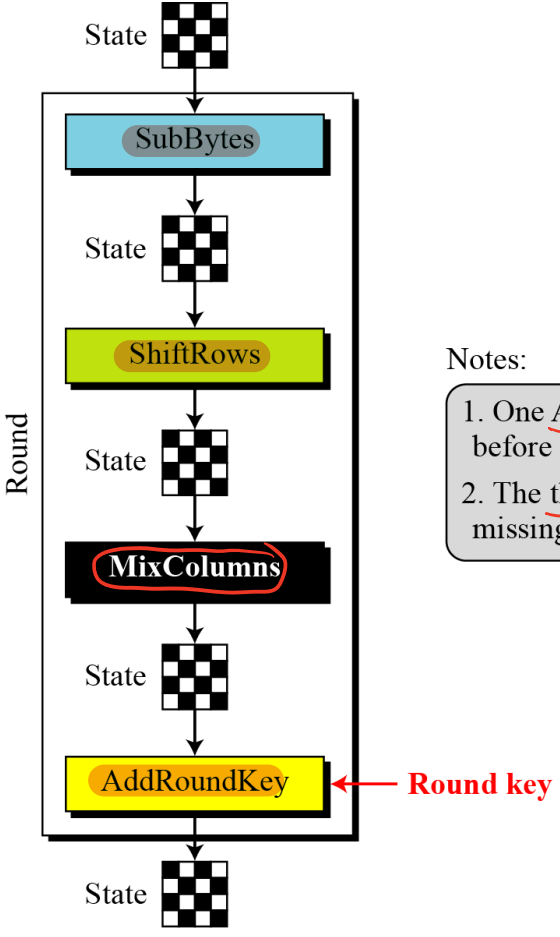
- Byte, Word, Block, State
  - 1word = 32bits = 4bytes, 1block = 128bits = 16 bytes
  - State is made of 4 × 4 bytes



Block-to-State and State-to-Block



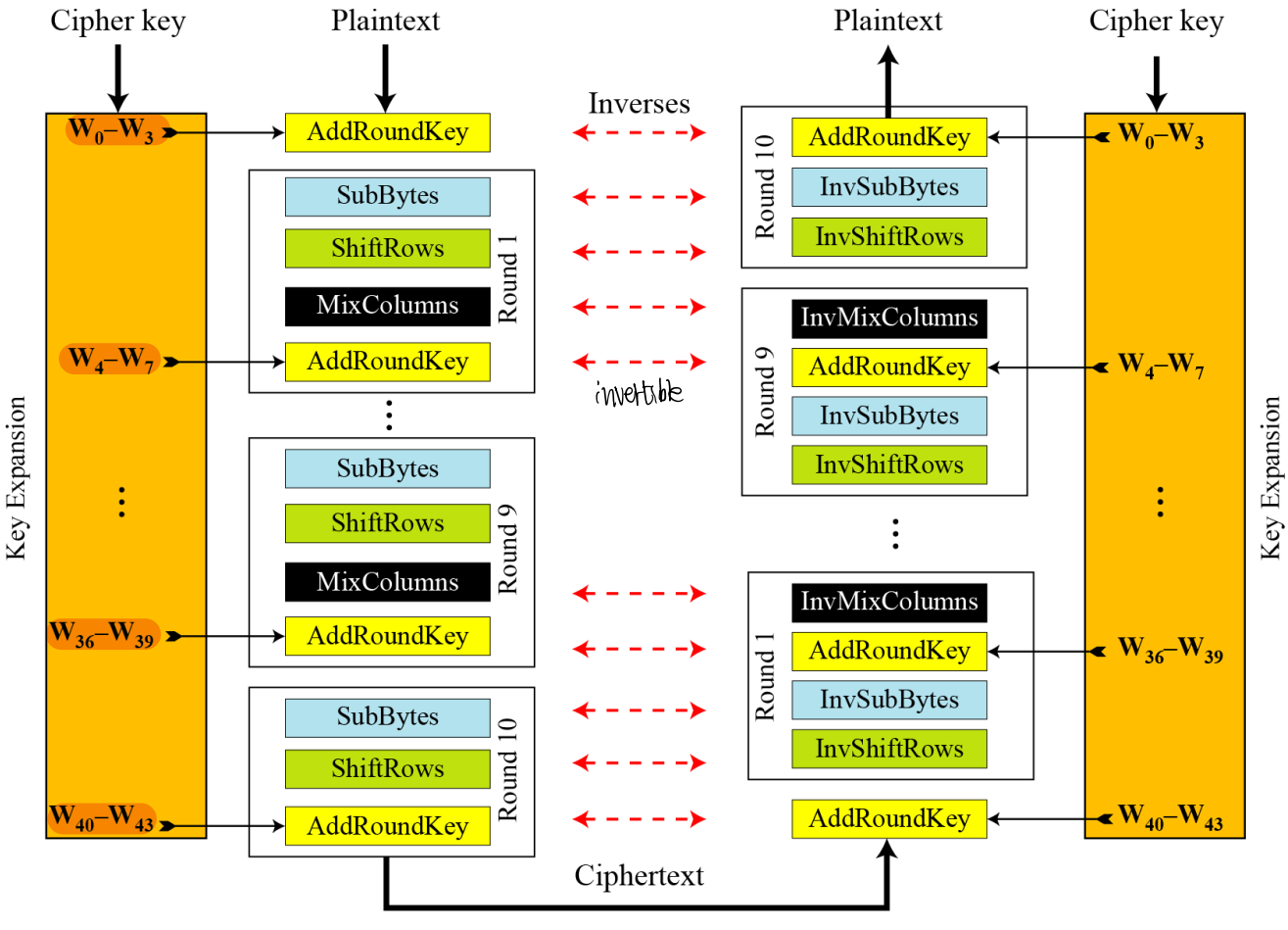
Structure of Each Encryption Round



Notes:

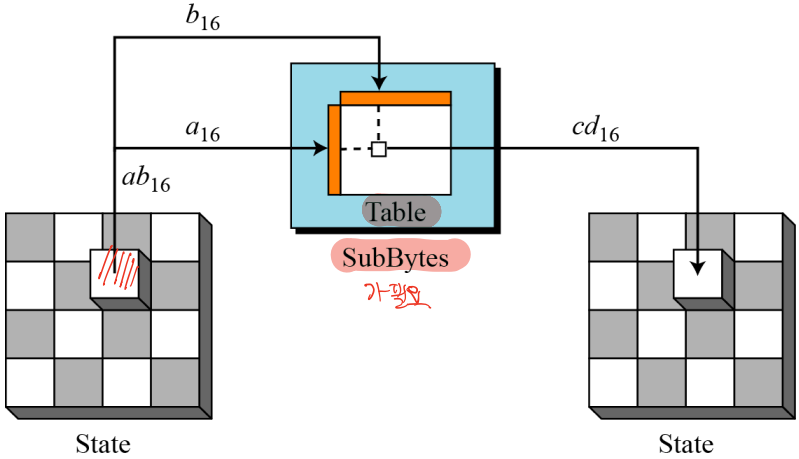
1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

AES-128 Schematic

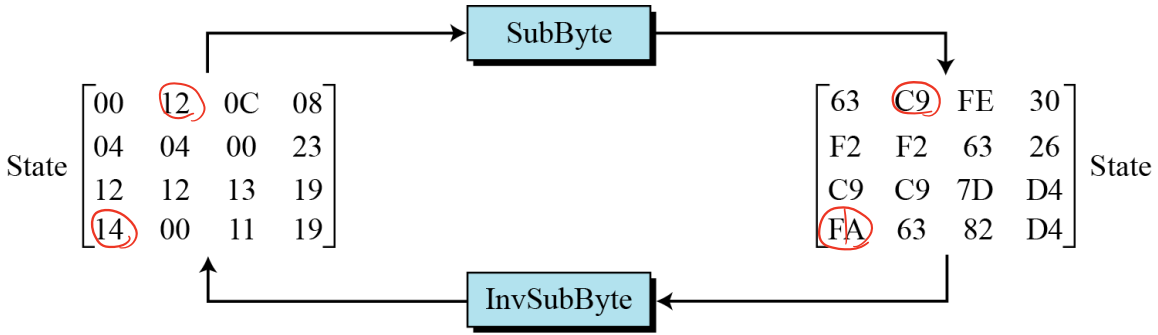


SubBytes – Substitution (1)

- SubBytes transformations
  - Involves 16 independent byte-to-byte transformations



  - Example:



SubBytes – Substitution (2)

- SubBytes transformation table (256 bytes)

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

SubBytes – Substitution (3)

- InvSubBytes transformation table (256 bytes)

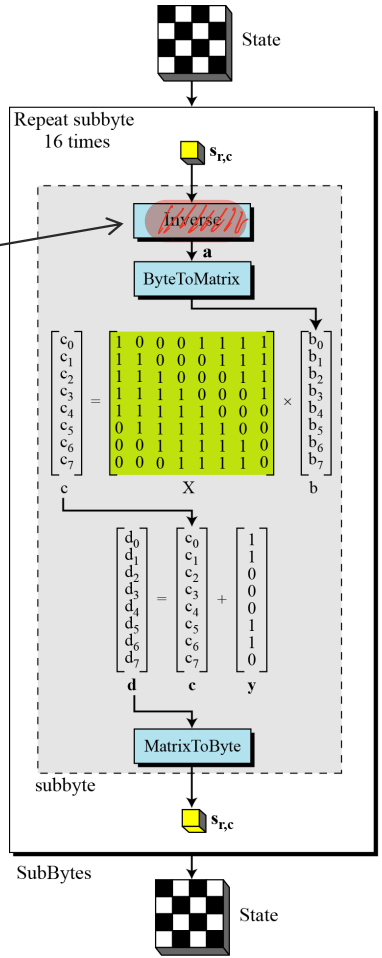
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

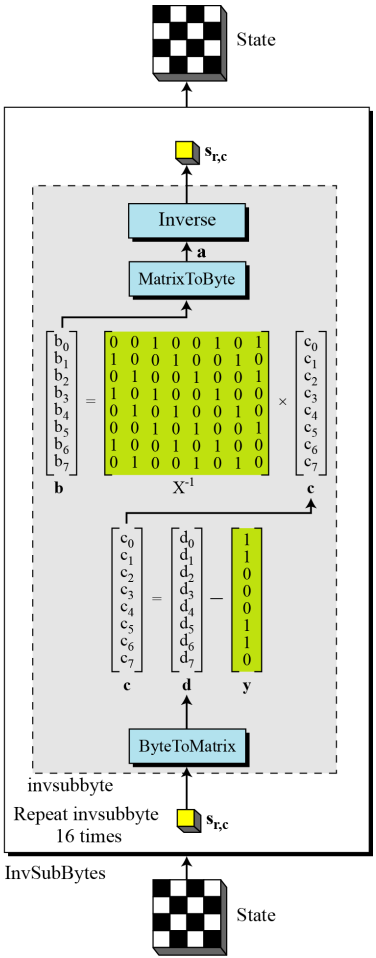
SubBytes – Substitution (4)

- SubByte transformation using the GF(2^8) field
  - the irreducible polynomial

    $= x^8 + x^4 + x^3 + x + 1$

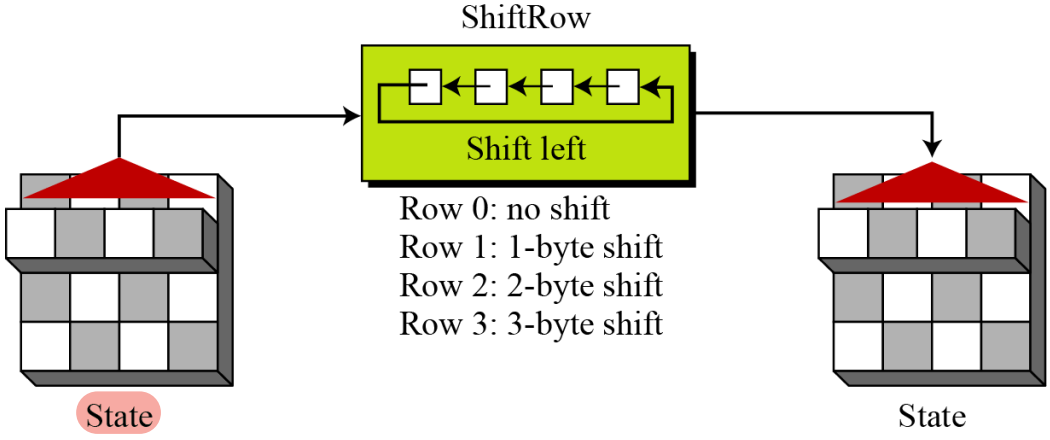
Nonlinearity !



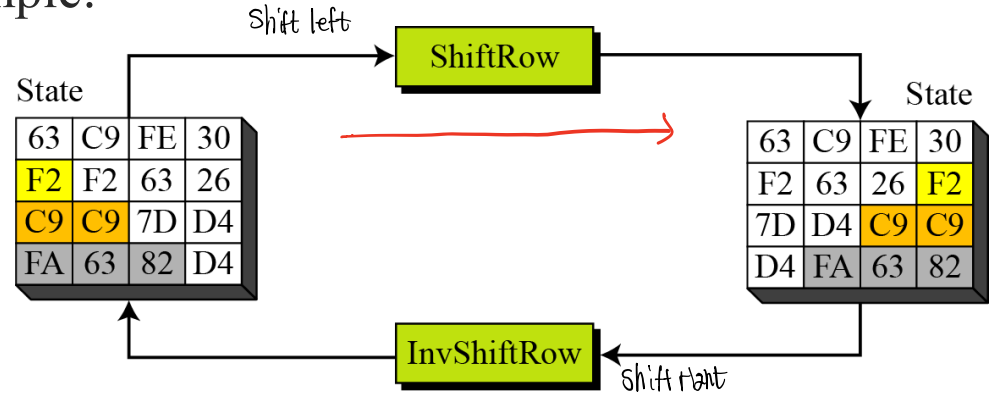


ShiftRows - Permutation

- The number of shifts depends on the row number

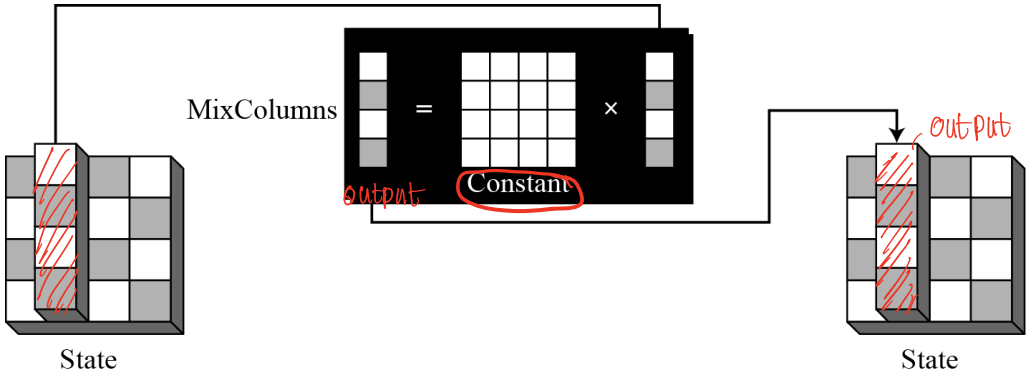


- Example:



MixColumns – Mixing (1)

- MixColumns transformation



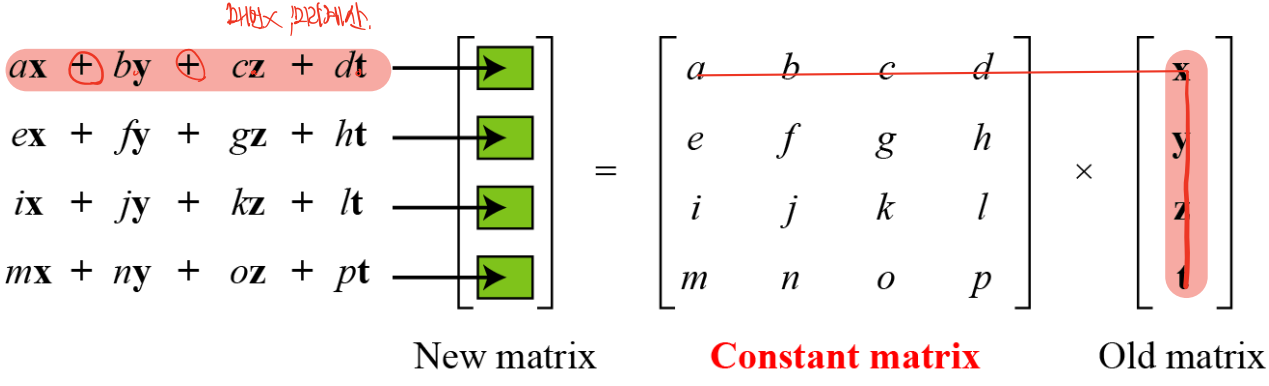
- Constant matrices

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \quad \text{Inverse} \quad \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

C $\qquad$ C^{-1}

MixColumns – Mixing (2)

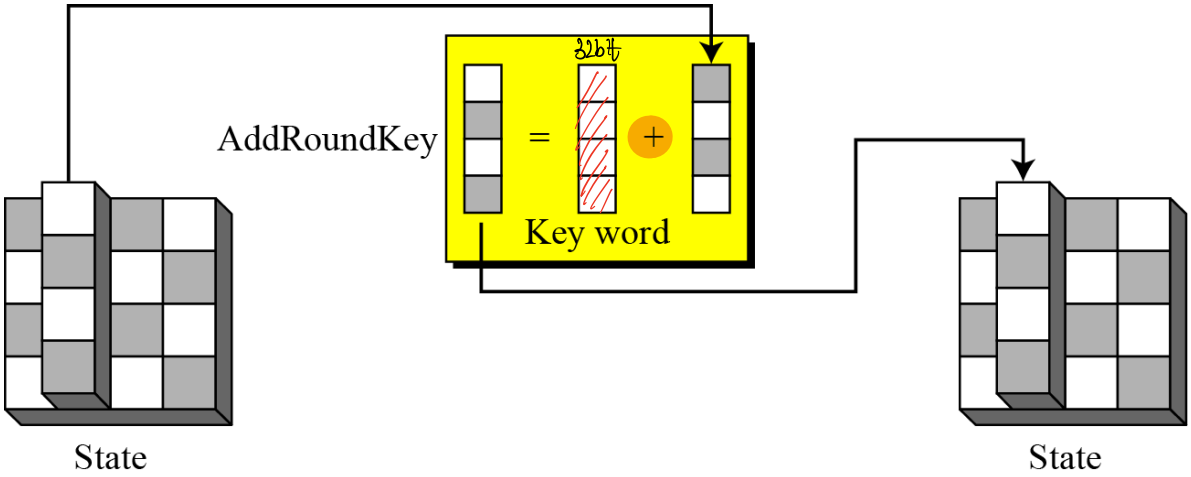
- Mixing bytes using matrix multiplication
  - Change bits, based on the bits inside the neighboring bytes (diffusion)



  - Operation (multiplication) is multiplication in GF(2^8)
  - Operation (addition) is XORing
    - To understand operations, need to know theory of polynomial-based fields
    - In practice, can make table (with array) for all possible entries

AddRoundKey

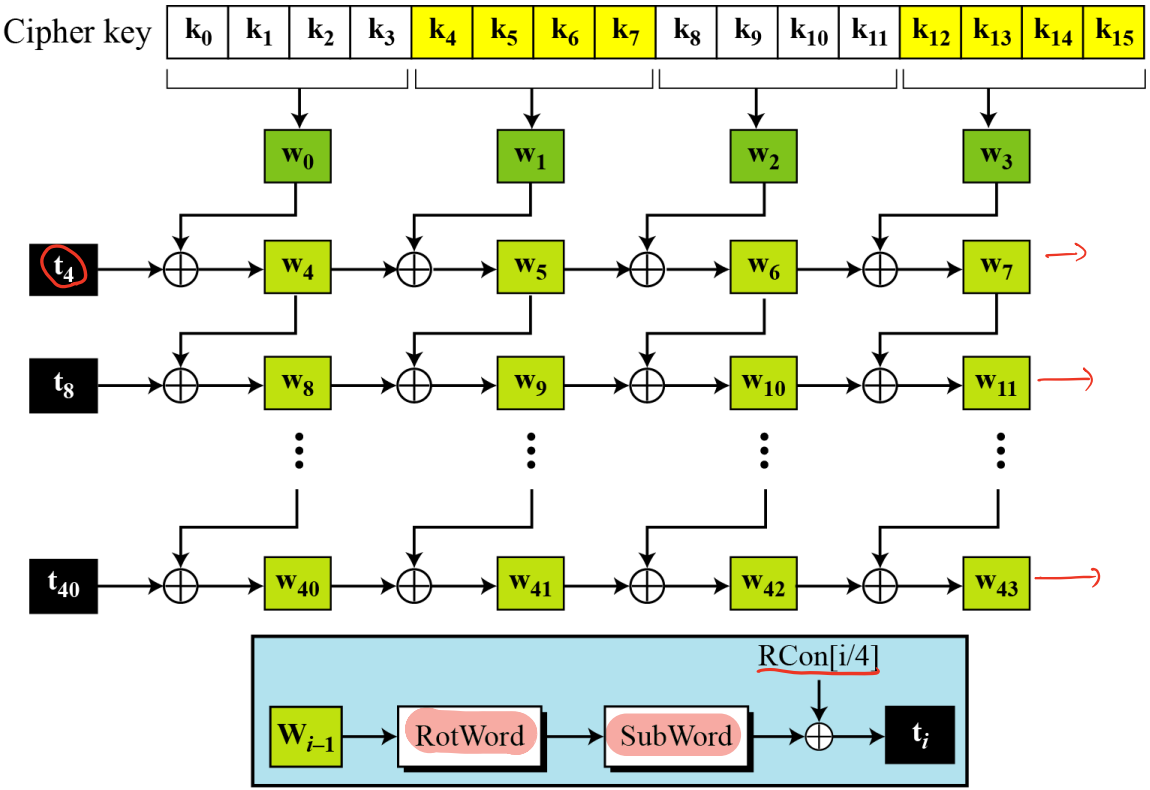
- AddRoundKey transformation



  - Operation(addition) is XORing
  - In each round, 4 key words (w_i, w_{i+1}, w_{i+2}, w_{i+3}) are needed
    - 1 word = 4 bytes = 32 bits

Key Expansion in AES-128 (1)

- Process in AES-128



Making of t_i (temporary) words $i = 4\ N_r$

Key Expansion in AES-128 (2)

- RotWord is similar to ShiftRows
  - Applied to only one row – 1 byte shift
- SubWord is similar to the SubBytes (applied to four bytes)
- Round constants (RCon)

Round	*Constant (RCon)*	*Round*	*Constant (RCon)*
1	(**01** 00 00 00)$_{16}$	6	(**20** 00 00 00)$_{16}$
2	(**02** 00 00 00)$_{16}$	7	(**40** 00 00 00)$_{16}$
3	(**04** 00 00 00)$_{16}$	8	(**80** 00 00 00)$_{16}$
4	(**08** 00 00 00)$_{16}$	9	(**1B** 00 00 00)$_{16}$
5	(**10** 00 00 00)$_{16}$	10	(**36** 00 00 00)$_{16}$

https://github.com/intel/tinycrypt

Encryption Results

- An encryption result

Plaintext:	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19
Cipher Key:	24	75	A2	B3	34	75	56	88	31	E2	12	00	13	AA	54	87
Ciphertext:	BC	02	8B	D3	E0	E3	B1	95	55	0D	6D	FB	E6	F1	82	41

- Avalanche effect

Plaintext 1:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Plaintext 2:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01
Ciphertext 1:	63	2C	D4	5E	5D	56	ED	B5	62	04	01	A0	AA	9C	2D	8D
Ciphertext 2:	26	F3	9B	BC	A1	9C	0F	B7	C7	2E	7E	30	63	92	73	13

- When using a key where all bits are 0s

Plaintext:	00	04	12	14	12	04	12	00	0c	00	13	11	08	23	19	19
Cipher Key:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Ciphertext:	5A	6F	4B	67	57	B7	A5	D2	C4	30	91	ED	64	9A	42	72

Performance

- AMD Opteron, 2.2 GHz (Linux)

	Cipher	Block/key size	Speed (MB/sec)
stream	RC4		126
	Salsa20/12		643
	Sosemanuk		727
block	3DES	64/168	13
	AES-128	128/128	109

인텔칩에서 오픈ssl 측정.

Homework 1

- https://github.com/intel/tinycrypt
  - AES encrypt/decrypt code analysis (due date: 10/7)

Q & A